Segment-Based Proxy Caching of Multimedia Streams

Kun-Lung Wu, Philip S. Yu and Joel L. Wolf
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
{klwu, psyu, jlw}@us.ibm.com

ABSTRACT

As streaming video and audio over the Internet becomes popular, proper proxy caching of large multimedia objects has become increasingly important. For a large media object, such as a 2-hour video, treating the whole video as a single web object for caching is not appropriate. In this paper, we present and evaluate a segment-based buffer management approach to proxy caching of large media streams. Blocks of a media stream received by a proxy server are grouped into variable-sized segments. The cache admission and replacement policies then attach different caching values to different segments, taking into account the segment distance from the start of the media. These caching policies give preferential treatments to the beginning segments. As such, users can quickly play back the media objects without much delay. Event-driven simulations are conducted to evaluate this segment-based proxy caching approach. The results show that (1) segment-based caching is effective not only in increasing byte-hit ratio (or reducing total traffic) but also in lowering the number of requests that require delayed starts; (2) segment-based caching is especially advantageous when the cache size is limited, when the set of hot media objects changes over time, when the media file size is large, and when many users may stop playing the media after only a few initial blocks.

Keywords

Video Caching, Proxy Caching, Segment-Based Caching, Variable-Sized Segmentation, Multimedia Streaming.

1. INTRODUCTION

The explosive growth of the World Wide Web has led to significant increases in user latency and network congestion for Internet applications. One popular approach to reducing response time and network traffic is to deploy proxy caches on the edge of the Internet close to the users. A proxy cache stores recently accessed web objects in the hope of satisfying future client requests without contacting the content server.

WWW10, May 1-5, 2001, Hong Kong.
ACM 1-58113-348-0/01/0005.

As requests for and delivery of streaming video and audio over the Web becomes more popular, caching of media objects on the edge of the Internet has become increasingly important. Recently, several commercial companies have announced media distribution services on the Internet using a number of proxy caches. Examples include Akamai (www.akamai.com), Digital Island (www.digisle.com), Enron (www.enron.com) and others. Companies that provide hardware and software caching products include Inktomi (www.inktomi.com), CacheFlow (www.cacheflow.com), Network Appliance (www.netapp.com) and others.

However, existing techniques for caching text and image objects are not appropriate for caching media streams. The main reason is due to the large sizes of typical media objects. For a large media file, such as a 2-hour video, treating the whole video as a single web object to be cached is impractical. Just storing the entire contents of several long streams would exhaust the capacity of a conventional proxy cache. Hence, only the very few video objects that are hot should be cached entirely. Most media objects probably should only be cached partially.

Because of the high start-up overhead and isochronous requirement, a streaming media request typically is not started by a proxy server until sufficient blocks of data are cached locally. Such delayed starts can frustrate users and make customers unhappy. To overcome this problem, the beginning portions of most media objects should be cached. Hence, from the caching perspective, the beginning portion of a media stream is more important than the later portion.

The importance of beginning portions of most media objects and the observation that most media objects should only be cached partially lead us to a segment-based approach to proxy caching of large media objects. Blocks of a media object received by the proxy server are grouped into variable-sized, distance-sensitive segments. In fact, the segment size increases exponentially from the beginning segment. For simplicity, the size of segment i is 2^{i-1} and contains media blocks $2^{i-1}, 2^{i-1}+1, \cdots, 2^i-1$. The motivation for such exponentially-sized segmentation, as compared with fixed size segmentation, is such that we can quickly discard a big chunk of a cached media object that was once hot but has since turned cold. This way the cache manager can quickly adjust to the changing reference patterns of partially cached objects. For example, the cache manager can release $1/2$ of a cached media object in a single action. In a fixed size segmentation, in contrast, a sequence of actions must be taken to achieve the same effect. This is particularly important for large media objects, such as videos.

The number of segments cached for each object is dynamically determined by the cache admission and replacement policies. They attach different caching values to different segments based on the reference frequency and the segment distance from the beginning of a media. The caching policies give preferential treatments to the beginning segments. Hence, a partially cached media object always starts from the beginning, facilitating immediate starts of user requests. In addition, we cache more segments for media objects with higher reference frequencies.

Event-driven simulations were conducted to evaluate this variable-sized, distance-sensitive segment approach to proxy caching of large media objects. We compared the segment-based proxy caching with a whole media approach and a prefix/suffix approach. In the prefix/suffix approach, a media is partitioned into a prefix and a suffix. A small portion of the cache is dedicated for caching the prefixes while the rest of the cache for the suffixes. We measured the byte-hit ratio as well as the percentage of requests with delayed starts. When a request arrives and the beginning blocks of the requested media are not cached, the request has a delayed start. Sensitivities analyses on various parameters were conducted. The results show that (1) the segment-based approach indeed is effective not only in increasing the byte-hit ratio over a wide range of conditions, but also in lowering the percentage of requests with delayed starts; and (2) the segment-based approach is especially advantageous under the conditions that (a) the cache size is limited, (b) the set of hot media objects changes over time, (c) the media file size is large, and (d) many users may decide to stop viewing the media playback after only a few initial blocks.

There has been research work on the multimedia caching and proxy services [1, 3, 9, 8, 10, 14, 13]. However, to the best of our knowledge, none has examined the effectiveness of variable-sized, distance-sensitive segment caching policies for large media objects. Earlier research on multimedia caching proposed storing a sliding interval of successive frames to satisfy client requests that arrive close together in time [5, 11]. Other recent work proposed having the proxy server cache a fixed subset of frames, such as the prefix of a stream or a subset of other frames, to reduce the overhead of transmitting to the client [9, 12, 7, 2, 6]. Various aspects of prefix caching were also studied, such as workahead smoothing [9, 8] and protocol considerations [3]. Unlike the segment-based caching approach presented in this paper, the suffix was not cached in [9]. Partial segment caching of media objects was proposed to be combined with a dynamic skyscraper broadcasting to reduce the media delivery cost from a remote server to regional servers [2, 4]. However, the emphasis in [2] was on caching the initial segments of many media objects and relying on remote multicast delivery of the later segments, rather than fully caching fewer highly popular objects. In contrast, in our segment-based caching scheme, the most popular media objects are fully cached while the less popular objects are partially cached. Moreover, the number of initial segments cached is dynamically determined by the popularity of an object.

The rest of the paper is organized as follows. Section 2 describes the details of the variable-sized, segment-based proxy caching, including the system architecture, media segmentation policy, and cache admission and replacement policies. Section 3 presents our event-driven simulations and results. Finally, Section 4 summarizes the paper.

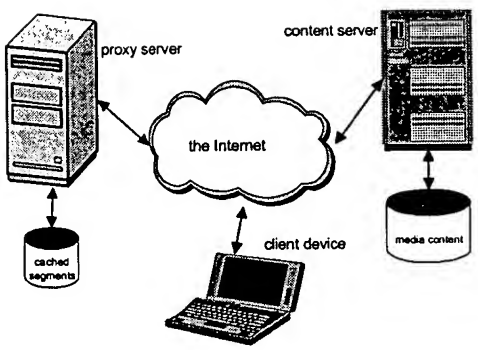

Figure 1: System architecture for media streaming on the Internet.

2. DESIGN OF SEGMENT-BASED PROXY CACHING OF MEDIA STREAMS

2.1 Caching media streams over the Internet

Fig.1 shows the general system architecture for media streaming over the Internet by employing proxy caching. A proxy server is placed geographically near the client access device, which can be a PC, a TV or another display device. A user request originated from a client device is directed to the proxy server. If the requested media object is cached, then it is transmitted right away to the client device. Otherwise, it is fetched first from the content server to the proxy server and then transmitted to the client device. It is assumed that the latency between a client device and the proxy server is negligibly small, but the latency between the proxy server and the content server is relatively large and cannot be ignored. Therefore, in order for a client device to immediately playing back the media, enough initial blocks must be present at the proxy server to mask the latency between it and the content server. Prefetching can be issued for the remaining not-yet-cached media.

2.2 Segmentation of media objects

To simplify the management of segments, we use a simple segmentation method (see Fig. 2 for an example). A media file is divided into multiple equal-sized blocks, which is the smallest unit of transfer. Multiple blocks are then grouped into a segment by the proxy server, where the cache admission and replacement policies attach different caching values to different segments. The size of a segment is sensitive to its distance from the beginning of the media. The closer to the beginning a segment is, the smaller the size it will be. The number of blocks grouped in segment i is 2^{i-1}. Segment i contains media blocks $2^{i-1}, 2^{i-1}+1, \cdots, 2^i-1$, if $i \geq 1$; Segment 0 contains block 0. In general, segment i is twice as large as segment $i-1$, except the last segment. The purpose of this exponentially-sized segmentation is to allow the cache manager to quickly discard a big chunk of cached media object, especially when the object size is large. For example, the cache manager can discard 1/2 of a cached object in a single action. In contrast, it takes a sequence of actions to do the same for a fixed size segmentation approach.

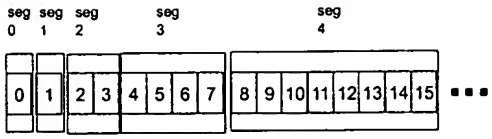

Figure 2: Example of media segmentation.

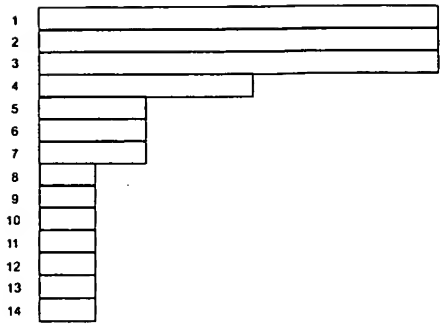

Figure 3: The size distribution of cached media objects for variable-sized segment scheme.

The segmentation process is transparent to the media content provider or the client device. It is the artifact introduced by the proxy server to make cache management more effective. The basic concept is to create smaller sized segments at the beginning and give them higher caching priorities. This is important because the initial segments determine the latency perceived by the users, and hence are more important to cache. Moreover, many viewers may decide to stop viewing after only a few initial segments. In such cases, it is not even necessary to fetch the later segments. The later segments are therefore made progressively larger so that we can reduce the number of segments that the proxy server needs to track and manage.

2.3 Cache admission policy

We consider two important cache management policies. One is the cache admission policy and the other is cache replacement policy. These two are closely related.

The primary idea of cache admission control is to permit only segments from media objects which are popular enough to enter the cache. The admission control applies different criteria to different segments of the same media object. The basic consideration is the distance of a segment from the beginning of the media object, i.e., the segment number. The beginning segments of an object have a critical impact on the initial delay to start the media. If cached, the video may be streamed immediately to the requesters. The later segments, if not cached, can be prefetched after the request is received. However, fetching these later segments does have a significantly negative impact on network traffic. Thus, these later segments still should be cached if they are requested frequently enough.

Thus, we use a two-tiered approach to admission control based on the segment number. For a segment with a segment number smaller than a threshold, K_{min}, it is always eligible for caching. However, for a segment with a segment number no smaller than K_{min}, it is determined eligible for caching only if its caching value is larger than some cached segments also with segment numbers no smaller than K_{min}. For simplicity, we assume that a portion of the cache capacity is used to store the initial segments while the rest to store the later segments. Hence, the initial K_{min} segments of a media object are cached as a unit and can only be replaced by the initial segments of other media objects. The value of K_{min} should be determined by the criterion that enough of the non-cached segments can be fetched in time from the content server so that continuous streaming can be guaranteed once it is started. It depends on the network delay between the proxy server and the content server. It also depends on the load condition of the content server.

With such a cache admission control, the highest numbered segment cached for a given media is always the last one to be included into the cache and the first one to be replaced. Each partially cached media object always gets a consecutive set of segments cached starting from the beginning. At least there are K_{min} initial segments stored for any cached objects. Some of the most frequently requested objects may have the entire contents cached. Fig. 3 shows an example of the size distribution of cached media objects. Some objects are fully cached, e.g., the entire contents of objects 1–3 are cached. Other objects are only partially cached. For example, objects 8–14 have the minimum K_{min} segments cached.

2.4 Cache replacement policy

The caching value of a segment depends on two variables: reference frequency of an object and the segment distance. We simply define a segment's caching value to be the ratio of reference frequency over segment distance. It is a simple reflection of the fact that we favor the initial segments of objects with higher reference frequencies. However, we did vary this value function in our simulations. We changed it to become the ratio of reference frequency over the n-th power of segment distance, where n is a positive number. But, the results were only slightly sensitive to n, if at all. Hence, $n = 1$ was used.

We use timestamp to estimate the reference frequency of an object. Each segment of an object uses the same timestamp. A timestamp records the last time an object is requested. Specifically, the reference frequency is estimated as the inverse of the time since the last reference to the current time. Namely, the reference frequency is estimated as $1/(T - T')$, where T is the current time and T' is the timestamp maintained for an object. If an object requested is not already in the cache, the timestamp is assumed to be a

minus infinite, so that the reference frequency will be zero. As a result, the caching value of segment i of an object is defined simply as $1/((T - T') \times i)$.

Two LRU stacks are maintained, one for the initial K_{min} segments and the other for the later segments. For each LRU entry, we maintain the object ID, the timestamp of the last request to this object, the last block ID of cached media, and the total number of requests currently still playing this object. Assume that K denote the first segment of media object O that is not yet cached, i.e., segments $0, 1, 2, \cdots, (K-1)$ of object O are cached. If $K \geq K_{min}$, this segment is eligible for caching only if its caching value is greater than that of its replacement segment. The candidate segments for replacement can be found from the LRU stack beginning from the bottom. Since we have always cached contiguous media from the beginning, a candidate segment for replacement is always the largest numbered segment. Thus, the replacement policy simply examines a small number of media objects from the bottom of the LRU stack to find the replacement segment.

Fig. 4 shows the caching algorithm for segment i of an object P using the admission as well as replacement policies outlined above. In order to maintain caching contiguous media for object P, we invoke this caching algorithm only if segment $i-1$ of object P has been cached. Otherwise, segment i will not be considered for caching. When an object is requested for the first time, the initial K_{min} segments are always eligible for caching and a simple LRU scheme can be used to find the replacement. However, since the reference frequency is zero, the later segments will not be eligible for caching. The later segments may be eligible for caching on subsequent requests. In order to make room for segment i of object P, many cached segments might be replaced. The least valued segment in the cache is identified one by one for replacement consideration. A replacement candidate segment will be removed from the cache if its caching value is smaller than that of segment i of object P and there is no active user currently playing back the media.

```
if (i < K_min) { // the LRU stack for initial segments
   find a replacement segments, if necessary;
   cache segment i;
}
else { // the LRU stack for later segments
   if (object P is referenced for the first time)
      exit;
   while ((there is not enough free space for segment i)
         and
         (replacement candidate can still be found)) {
      find object Q whose largest cached segment is least
      valued;
      let segment j be the largest cached segment of
      object Q;
      if ((1/((T - T'_P) × i) > 1/((T - T'_Q) × j)) and
         (no user is playing Q))
         replace segment j of object Q and increase
         free space;
   }
   if (there is enough free space for segment i of
   object P)
      cache segment i;
}
```

Figure 4: Caching algorithm for segment i of object P.

3. PERFORMANCE EVALUATION

3.1 Methodology

We implemented an event-driven simulator which models a proxy cache server to evaluate this variable-sized segment-based approach. Two LRU stacks were implemented to track media objects in the cache. One was to track the initial segments and the other to track the later segments. A portion C_{init} of the total cache capacity was dedicated for the initial segments. The later segments are managed in another LRU stack, but segment-based cache admission and replacement policies (see Fig. 4) are used instead of simply LRU. We computed two important performance metrics: byte-hit ratio and fraction of requests with delayed starts. The byte-hit ratio measures the ratio of total bytes from cached objects over the total bytes of objects requested. When a request arrives and the initial K_{min} segments are not in the proxy cache, it has a delayed start.

For the simulations, we assumed that the media objects are videos. The video size is uniformly distributed between $0.5B$ and $1.5B$ blocks, where B is the mean video size. The default value of B is 2,000. The playing time for a block is assumed to be 1.8 seconds. In other words, the playing time for a video is between 1,800 seconds and 5,400 seconds, or 30-90 minutes. The cache size is expressed in terms of number of media blocks. The default cache size is 400,000 blocks. The inter-arrival time is assumed to be exponentially distributed with mean λ. The default value of λ is 60.0 seconds. Table 1 shows the definition of these system parameters and the default values used in the simulations.

The requested video titles are drawing from a total of M distinct video titles. The popularity of each of the M video titles follows a Zipf-like distribution Zipf(x, M) [15]. A Zipf-like distribution takes two parameters, x and M, the former corresponding to the degree of skew. The distribution is given by $p_i = c/i^{1-x}$ for each $i \in \{1, \cdots, M\}$, where $c = 1/[\sum_{i=1}^{M} 1/i^{1-x}]$ is a normalization constant. Setting $x = 0$ corresponds to a pure Zipf distribution, which is highly skewed. On the other hand, setting $x = 1$ corresponds to a uniform distribution with no skew. The default value for x is 0.2 and that for M is 2,000.

The popularity of each video title changes over time. This is used to simulate the scenario that there may be different user groups accessing the video titles at different times and their interests may be different. In other words, the most popular videos at the moment may be replaced by another ones at a later time. In our simulations, the popularity distribution changed every R requests. When it did, another well-correlated Zipf-like distribution with the same parameters was used [13]. The *correlation* between two Zipf-like distributions is modeled by using a single parameter k that can take on any integer value between 1 and M. First, the most popular video in Zipf-like distribution 1 is made to correspond to the r_1-th most popular video in Zipf-like distribution 2, where r_1 is chosen randomly between 1 and k. Then, the second most popular video in distribution 1 is made to correspond to the r_2-th most popular video in distribution 2, where r_2 is chosen randomly between 1 and $min(M, k+1)$, except that r_1 is not allowed, and so on. Thus, k represents

Table 1: System parameters and default values

Notation	Definition (Default values)
B	mean number of blocks per video (2,000 blocks)
λ	mean request inter-arrival time (60 s)
C	total cache capacity (400,000 blocks)
C_{init}	portion of cache capacity used to cache initial segments (10%)
K_{min}	initial segments cached for a video (6 segments, or 32 blocks)
M	number of distinct video titles (2,000)
Zipf(x, M)	Zipf-like distribution for video titles (Zipf(0.2, 2,000))
k	maximum shifting distance for a hot video (10)
R	number of requests between shifting of hot videos (200)

the maximum position in popularity a video title may shift from one distribution to the next. $k = 1$ corresponds to perfect correlation, and $k = M$ to the random case or no correlation. We used an indirect video mapping array to implement this change of popularity distribution. The first element in the mapping array always represents the most popular. However, the video to which the first element maps may change. Hence, the change in popularity distribution was implemented by changing the videos to which the array maps.

3.2 Simulation results

We compared the variable-sized segment approach to a full video and a prefix/suffix schemes. The full video scheme simply uses an LRU for replacement. Every requested video is cached in its entirety. The prefix/suffix scheme partitions a video into a prefix and a suffix. The prefix size is the same as the K_{min} initial segments of the variable-sized segment approach. Furthermore, the same portion of cache capacity is dedicated to store the prefixes. Both prefix and suffix are managed using LRU replacement. Note that an object is always cached once it is referenced in an LRU policy. In contrast, there is a cache admission policy in the variable-sized segment approach.

3.2.1 The impact of cache size

First, we studied the impacts of cache size on the byte-hit ratio and delayed start. For a wide range of cache sizes, the variable-sized segment approach has the highest byte-hit ratio and the lowest fraction of requests with delayed starts. Fig. 5 shows the impact of cache size on the byte-hit ratio. Fig. 6 presents the impact of cache size on the fraction of requests with delayed starts. The full video approach and the prefix/suffix has comparable byte-hit ratio (see 5), with the full video approach having a slight advantage over the prefix/suffix one. The advantage in byte-hit ratio of the variable-sized segment approach is more significant for a smaller cache size. For instance, the byte-hit ratio for the variable-sized segment approach is 21% better for a smaller cache size of 300,000 and 8% better for a much larger cache size of 900,000. With higher byte-hit ratio, the variable-sized

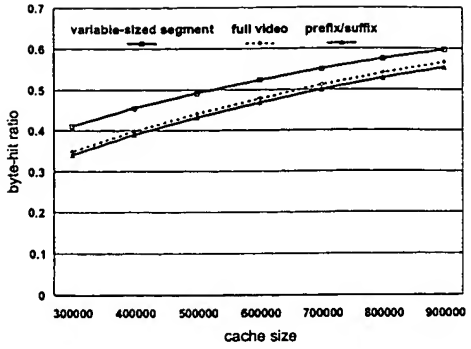

Figure 5: The impact of cache size on byte-hit ratio.

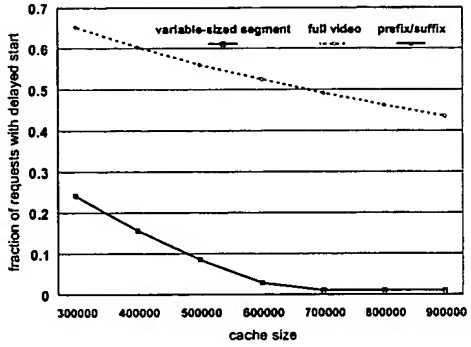

Figure 6: The impact of cache size on delayed start (variable-sized segment and prefix/suffix are identical).

segment approach can achieve the same caching effectiveness with a smaller storage requirement. As a result, it can reduce the system cost for the organizations running proxy caching. For example, in order to achieve a 50% byte-hit ratio, the variable-sized segment approach needs only 500,000 blocks in cache size while the prefix/suffix approach requires 700,000 blocks, a 40% increase.

Even though the full video and the prefix/suffix approaches perform almost equally in byte-hit ratio, they differ dramatically in the fraction of requests with delayed starts (see Fig. 6). The full video approach has a significantly higher fraction of requests with delayed starts. For example, for a cache size of 400,000 blocks, 60% of the requests cannot start immediately under the full video approach. On the other hand, only 15.6% of requests need to be delayed. Because of the same amount of cache capacity dedicated for storing the initial blocks, the variable-sized segment and the prefix/suffix approaches perform identically in Fig. 6 for the whole range of cache sizes.

3.2.2 The impact of skew in video popularity

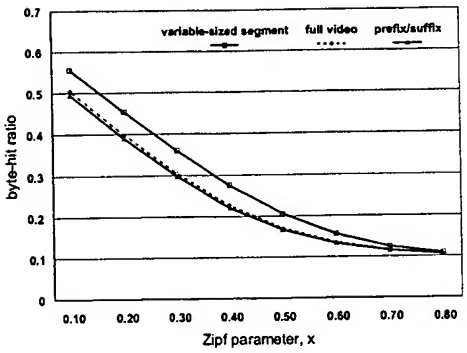

Figure 7: The impact of skew in video popularity on byte-hit ratio.

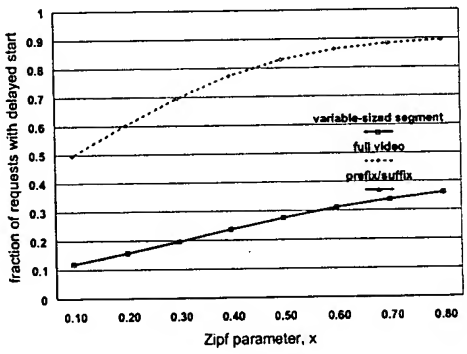

Figure 8: The impact of skew in video popularity on delayed start (variable-sized segment and prefix/suffix are identical).

Secondly, we examined the impact of skew in video popularity on the byte-hit ratio and delayed start. Again, the variable-sized segment approach has the highest byte-hit ratio and lowest fraction of requests with delayed starts under a wide range of degrees of skew in video popularity. Specifically, we varied the Zipf parameter, x, from 0.1 to 0.8. Fig. 7 shows the impact of skew in video popularity on byte-hit ratio while Fig. 8 shows the impact on delayed start. In general, the more skewed the video popularity is, i.e., more viewers are interested in fewer titles, the better the byte-hit ratio is. Thus, caching is more effective if most viewers are repeatedly requesting the same smaller number of videos. For example, for $x = 0.1$, the byte-hit ratios are above 50% for all three approaches. However, for $x = 0.8$, the byte-hit ratios are only about 10% for all three approaches. Under such conditions, caching is simply not effective, no matter which approach is employed.

In addition to Zipf parameter, x, we also studied the impact of the maximum video shifting position k, during popularity distribution change. The default R was set to be 200.

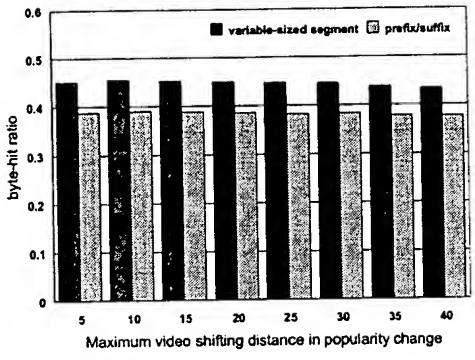

Figure 9: The impact of the maximum shifting position of a video.

Fig. 9 shows the impact of the maximum shifting position of a video. Here, we shows the variable-sized segment and the prefix/suffix approaches. In general, as the maximum distance increases, the byte-hit ratios for both approaches decrease, but only very slightly. The variable-sized segment approach is consistently better than the prefix/suffix approach. As expected, this is due to the fact that the popularity distributions in video titles are highly correlated with k ranging from 5 to 40. Note that $k = 1$ represents perfect correlation and $k = M = 2,000$ represents no correlation. We also varied the values of R, and the results were similar in the sense that the byte-hit ratios are only slightly sensitive to R for the similar reason.

3.2.3 The impacts of other system parameters

Fig. 10 shows the impact of mean video length on the byte-hit ratio. In general, as the media file size increases, the byte-hit ratios decrease for all three approaches. Notice that the advantage of the variable-sized segment approach over the other two is more significant when the media file size is larger. For example, the advantage is about 28% for mean video size of 3,000 blocks, but is only about 9% for the case of 1,000 blocks. This shows that the variable-sized segment approach is particularly useful in proxy caching of large media streams.

Besides the media file size, the number of distinct media objects can also impact the caching effectiveness. Generally, there are many media objects exist on the Web. As user requests spread over more distinct objects, caching becomes less effective. Fig. 11 shows the byte-hit ratios of the three approaches under different numbers of distinct objects users can request. Once again, the variable-sized segment approach has a bigger advantage over the other two when the condition for caching is less favorable.

Fig. 12 and Fig. 13 examine the percentage of cache capacity dedicated for storing the initial segments or prefixes. Because of reduced cache capacity for the later segments or suffixes, the byte-hit ratio decreases as the percentage used for initial segments increases. This slight decrease in byte-hit ratio can be offset by the substantially increased benefits of reduced delayed starts. For example, let us compare the cases of 5% and 15%. The byte-hit ratio is barely decreased,

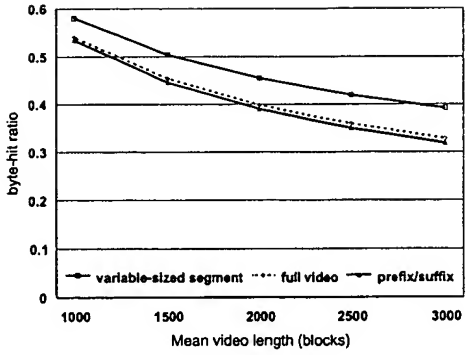

Figure 10: The impact of video length.

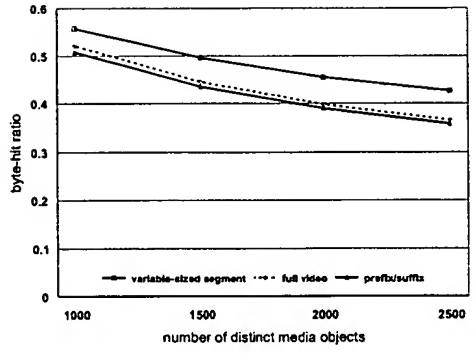

Figure 11: The impact of total number of distinct media objects.

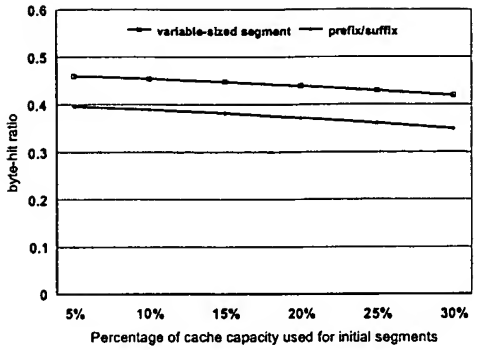

Figure 12: The impact of cache capacity used for initial segments on byte-hit ratio.

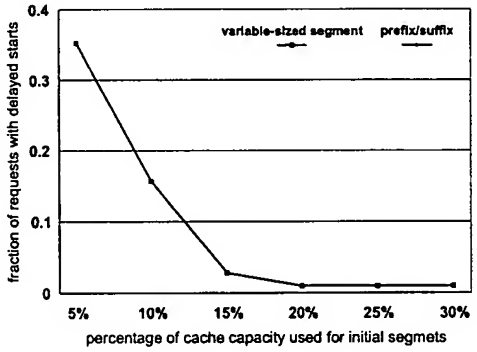

Figure 13: The impact of cache capacity used for initial segments on delayed starts.

but the fraction of delayed starts drops substantially. However, no more benefits can be derived once the percentage of cache for storing the initial segments increases beyond 20%.

3.2.4 The impact of user viewing behavior

Finally, we study the impact of a not uncommon user behavior on the Web. Most users may prematurely stop playing back a video after viewing only the first few segments. There are many possible reasons for a user to stop playing prematurely. Among them, the media content may not be what the user has anticipated or the media is simply too long and the user has lost interest in it. For this study, we created three scenarios representing partial completion of viewing the video. In partial completion scenario I, there are 50% of users completing the entire video while 50% of users stopping at half of the video. In partial completion scenario II, 25% of users complete 1/4 of a video, 25% of them complete 1/2 of a video, 25% of them complete 3/4 of a video and another 25% of them complete the entire video. In partial completion scenario III, 50% of users complete 1/4, 20% of them complete 1/2, 20% of them complete 3/4

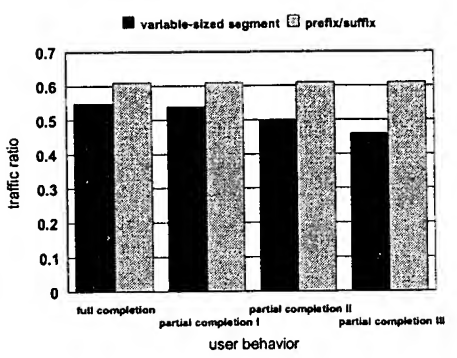

Figure 14: The impact of user viewing behavior on traffic ratio.

and only 10% of users complete the entire video. Partial scenario III represents a case where most viewers stops quite early.

Fig. 14 shows the traffic ratio of both the variable-sized segment and prefix/suffix approaches. The traffic ratio simply computes the ratio of total bytes retrieved from the content server over the total bytes for requested objects. For the variable-sized segment approach, the prefetch request for the next segment is issued when the proxy server is transmitting the first blocks of the current segment. Namely, we only prefetch one segment ahead. Hence, if a user stops early, network traffic may be saved. For the prefix/suffix approach, however, the prefetch request for the suffix must be issued once a request is made, making it unable to save traffic even if a user prematurely stops viewing a video. Here we assumed that the entire suffix was fetched. In Fig. 14, the traffic ratio decreases for the variable-sized segment approach as more users stop viewing the video earlier. Since we always prefetch the next segment, the traffic ratio of partial completion scenario I is the same as the full completion case.

4. SUMMARY

In this paper, we have presented a variable-sized segment approach to proxy caching of large media objects, such as videos. Proper proxy caching is very important as streaming video and audio over the Internet becomes ever more popular. Instead of treating the entire video as a web object, our segment-based approach groups media blocks into variable-sized segments with a simple segmentation method. The cache admission and replacement policies assign different caching values to different segments, taking into account both reference frequency and segment distance from the beginning of the media. These caching policies give preferential treatments to the initial segments, resulting partially cached objects starting from the beginning segment.

Event-driven simulations were conducted to evaluate the variable-sized segment approach and compare it with a full video approach and a prefix/suffix caching approach. The prefix/suffix approach partitions a video into a prefix and a suffix. The results show that (1) indeed the variable-sized segment approach is effective in not only increasing the byte-hit ratio (or reducing total traffic) but also lowering the the fraction of requests that require delayed starts; (2) variable-sized segment approach is particularly effective when the cache size is limited, when the set of hot media objects changes over time, when requests spread over a large number of media objects, when the media file size is large and when many users may stop viewing the video after only a few initial segments.

5. REFERENCES

[1] M. Y.M. Chiu and K.-H. A. Yeung. Partial video sequence caching scheme for VOD systems with heterogeneous clients. *IEEE Trans. on Industrial Electronics*, 45(1):44–51, Feb. 1998.

[2] D. L. Eager, M. C. Ferris, and M. K. Vernon. Optimized regional caching for on-demand data delivery. In *Proc. of Multimedia Computing and Networking*, Jan. 1999.

[3] S. Gruber, J. Rexford, and A. Basso. Protocol considerations for a prefix-caching proxy for multimedia streams. *Computer Network*, 33(1-6):657–668, June 2000.

[4] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proc. of ACM SIGCOMM 97 Conference*, pages 89–100, Sept. 1997.

[5] M. Kamath, K. Ramamritham, and D. Towsley. Continuous media sharing in multimedia database systems. In *Proc. Int. Conf. on Database Systems for Advanced Applications*, Apr. 1995.

[6] Z. Miao and A. Ortega. Proxy caching for efficient video services over the Internet. In *Proc. of Packet Video Workshop*, Apr. 1999.

[7] R. Rejaie, M. Handley, H. Yu, and D. Estrin. Proxy caching mechanism for multimedia playback streams in the Internet. In *Proc. of Int. Web Caching Workshop*, Mar. 1999.

[8] J. D. Salehi, Z.-L. Zhang, J. Kurose, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. *IEEE Trans. on Networking*, 6(4):397–410, Aug. 1998.

[9] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proc. of IEEE INFOCOM*, Mar. 1999.

[10] J. Shim, P. Scheuermann, and R. Vingralek. Proxy cache algorithms: Design, implementation, and performance. *IEEE Trans. on Knowledge and Data Engineering*, 11(4):549–562, Jul/Aug 1999.

[11] R. Tewari, H. M. Vin, A. Dan, and D. Sitaram. Resource-based caching for web servers. In *Proc. of SPIE/ACM Conf. on Multimedia Computing and Networking*, Jan. 1998.

[12] Y. Wang, Z.-L. Zhang, D. Du, and D. Su. A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers. In *Proc. of IEEE INFOCOM*, pages 660–667, Apr. 1998.

[13] J. L. Wolf, P. S. Yu, and H. Shachnai. Disk load balancing for video-on-demand systems. *Multimedia Systems*, 5:358–370, 1997.

[14] K.-L. Wu and P. S. Yu. Latency-sensitive hashing for collaborative web caching. *Computer Network*, 33(1-6):633–644, June 2000.

[15] G. K. Zipf. *Human Behaviour and the Principles of Least Effort*. Addison-Wesley, Cambridge, MA, 1949.

Vitae

Kun-Lung Wu received the B.S. degree in Electrical Engineering from the National Taiwan University, Taipei, Taiwan and the M.S. and Ph.D. degrees in Computer Science from the University of Illinois at Urbana-Champaign. Since 1990, he has been with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, where he is a member of the Software Tools and Techniques Group. His current research interests include infrastructure design issues of the WWW, personalization and data mining tools for the WWW, Internet applications and pervasive computing. He has published extensively in the areas of Web caching, database performance, disk subsystems, transaction and query processing, multimedia systems, mobile computing and reliable computing. Dr. Wu has published more than 50 papers in refereed journals and conferences and over 30 research reports. He holds or has applied for 17 US patents.

Dr. Wu is a member of the ACM and the IEEE Computer Society. He is currently on the editorial board of the IEEE Trans. on Knowledge and Data Engineering, serving as an associate editor. He is the general chair for the 3nd International Workshop on E-Commerce and Web-Based Information Systems (WECWIS 2001). He has served as an organizing and program committee member on various conferences. He has received various IBM awards, including Research Division Award and Invention Achievement Awards.

Philip S. Yu received the B.S. Degree in E.E. from National Taiwan University, Taipei, Taiwan, the M.S. and Ph.D. degrees in E.E. from Stanford University, and the M.B.A. degree from New York University. He is with the IBM Thomas J. Watson Research Center and currently manager of the Software Tools and Techniques group. His research interests include data mining, Internet applications and technologies, database systems, multimedia systems, parallel and distributed processing, disk arrays, computer architecture, performance modeling and workload analysis. Dr. Yu has published more than 270 papers in refereed journals and conferences. He holds or has applied for 184 US patents.

Dr. Yu is a Fellow of the ACM and a Fellow of the IEEE. He is the Editor-in-Chief of IEEE Transactions on Knowledge and Data Engineering. He is also an associate editor of ACM Transactions on the Internet Technology and that of Knowledge and Information Systems. He is a member of the IEEE Data Engineering steering committee and is also on the steering committee of IEEE Conference on Data Mining. In addition to serving as program committee member on various conferences, he was the program co-chair of the 11th Intl. Conference on Data Engineering and the program chairs of the 2nd Intl. Workshop on Research Issues on Data Engineering: Transaction and Query Processing, the PAKDD Workshop on Knowledge Discovery from Advanced Databases, and the 2nd Intl. Workshop on Advanced Issues of E-Commerce and Web-based Information Systems. He served as the general chair of the 14th Intl. Conference on Data Engineering. He has received several IBM and external honors including Best Paper Award, 2 IBM Outstanding Innovation Awards, an Outstanding Technical Achievement Award, 2 Research Division Awards and 53rd plateau of Invention Achievement Awards. He also received an IEEE Region 1 Award for "promoting and perpetuating numerous new electrical engineering concepts" in 1999. Dr. Yu is an IBM Master Inventor and was recognized as one of the IBM's ten top leading inventors in 1999.

Joel L. Wolf received his Ph.D. from Brown University in 1973 and his Sc.B. from MIT in 1968, both in mathematics. He is currently a staff member at the IBM T.J. Watson Research Center, with interests in mathematical optimization. He has won 2 IBM Outstanding Innovation Awards, is a Master Inventor at IBM, and a Fellow of the IEEE. He has also been an Assistant Professor of Mathematics at Harvard University, as well as a Distinguished Member of Technical Staff and manager at Bell Laboratories.

THIS PAGE BLANK (USPTO)